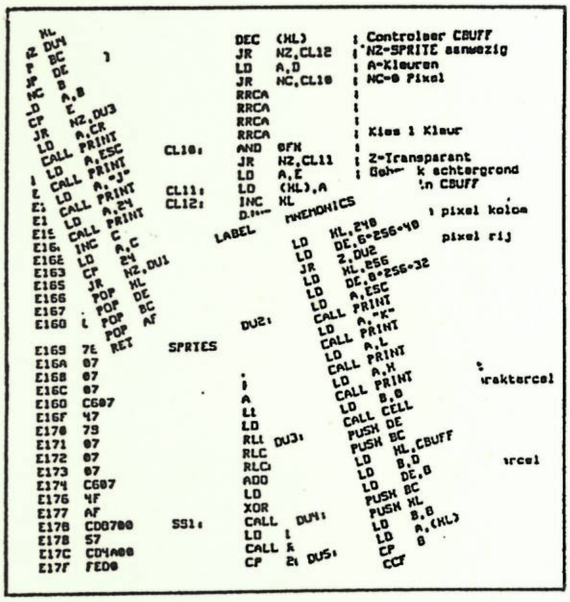
RF ASSEMBLER

MSX 2



R. Fokkens

Assembler / Disassembler / Debugger

onder MSX-DOS

Handleiding

WAARSCHUWING

Het is niet toegestaan de software en handleiding in biliotheken op te nemen, met de bedoeling deze uit te lenen of te verhuren, zonder voorafgaande schriftelijke toestemming van de uitgever.

Een uitgave van: Terminal Software Publicaties
Postbus 111, 5110 AC Baarle Nassau

1e druk november 1987 (versie 1.1)
2e druk april 1988 (versie 2.1)

ISBN 90-6883-043-0

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Fokkens, Rolf

RF-Assembler voor MSX2 / Rolf Fokkens. -
Baarle Nassau : Terminal Software Publicaties
Met diskette.
ISBN 90-6883-043-0 losbl.
SISO 525.6 UDC 681.3.068
Trefw.: RF-Assembler (computerprogramma).

Dit pakket wordt slechts dan door de uitgever gegarandeerd en in de toekomst van updates voorzien, indien de koper van dit pakket de registratiekaart heeft ingestuurd en de koper door de uitgever is geregistreerd als gebruiker.

INDEX

Op de Diskette (SSDD) treft U het volgende aan:

RFASMV21.COM	(De assembler)
DIRSORT.ASM	(source-file van een gesorteerde disk-directory)

INLEIDING

RF-ASSEMBLER is een krachtige, onder MSX-DOS functionerende assembler voor MSX2 computers. RF-ASSEMBLER wijkt voornamelijk van andere assemblers af in de volgende aspecten:

- RF-ASSEMBLER is een assembler, disassembler en debugger in een. Dankzij deze eigenschap kan men snel programma's ontwikkelen.
- RF-ASSEMBLER slaat de source bijzonder efficiënt op. Eigenlijk lijkt de source meer op het vertaalde programma dan op tekst zoals die wordt ingetypt. Gemiddeld zal dit al gauw een factor 2 uitmaken. Een voorbeeld: LD A,B wordt opgeslagen als slechts een byte: 78H. Het voert te ver om hierover uit te weiden, maar het is eenvoudig om dit met het MD-commando (zie verderop) zelf te onderzoeken. Door deze compacte opslag gaat ook het assembleren zeer snel.
- Als de computer is uitgerust met een MEMORY MAPPER, wordt de opslag van de source extra efficiënt, omdat de labels in een tabel in een apart stuk geheugen worden opgeslagen.

De twee laatst genoemde voordelen gelden ook als de source op disk staat: dus ook wordt de ruimte op disk niet verkwist!

RF-ASSEMBLER versie 2.1 is een uitbreiding van versie 1.1. hoewel het programma nu groter is geworden, zult u geen problemen krijgen met het feit dat er minder geheugen over is voor de source, want de opslag is ook iets efficiënter geworden. De belangrijkste verbeteringen:

- Het rekenen met labels en constanten in de operanden is uitgebreid.
- Breakpointing is aan de debugger toegevoegd.
- Het RG-commando is uitgebreid: men kan nu alle registers apart een waarde geven, met name de inhoud van het F-register kan nu eenvoudig worden veranderd.
- De error meldingen bij disk I/O worden nu door RF-ASSEMBLER opgevangen, zodat men niet meer per ongeluk RF-ASSEMBLER kan verlaten door in zo'n geval te kiezen voor ABORT.

Verder zijn de volgende commando's toegevoegd:

- AI : Assemble Immediately
- BK : BreaKpoint
- CB : Clear Breakpoint
- CV : ConVert
- GO : Go

Deze commando's worden verderop in de handleiding toegelicht.

RF-ASSEMBLER is voor het grootste deel ontwikkeld met behulp van zichzelf, daardoor is RF-ASSEMBLER in ruime mate getest. mocht U echter toch nog fouten ontdekken, dan vindt u achterin dit mapje een garantiekaart. Zodoende bent u verzekerd van een correct werkend exemplaar.

April 1988,
R. Fokkens

ALGEMEEN

Om te leren hoe de assembler werkt is het van belang dat U eerst met de assembler-commando's kunt omgaan. Ze worden in hoofdstuk 1 uitgelegd.

STARTEN

Het starten van RF-ASSEMBLER gebeurt (onder MSX-DOS) als volgt:

U tikt in:

RFASMV21 gevolgd door <RETURN>

Zodra U RF-ASSEMBLER gestart heeft, verschijnt er:

```
published by: Terminal Software Publicaties (c)1987


RF-ASSEMBLER for MSX-2 version 2.1 - No. xxxx
By Rolf Fokkens - 1987

RAM: Memory mapper (*)

Free mem : xxxxx (xxxx-xxxx)
# Labels : xxxx  (Max:xxxx )
?
```

Zoals al vermeld is in de inleiding: RF-ASSEMBLER werkt zowel met als zonder MEMORY MAPPER. De assembler herkent zelf de indeling van het geheugen. Als na 'RAM:' (zie *) vermeld wordt:

- Memory Mapper dan is Uw computer uitgerust met minimaal 128K RAM en een memory mapper.
- 64KByte dan beschikt Uw MSX2 over 64K RAM of heeft geen memory mapper. Dit laatste is het geval bij de Philips VG 8230.

Het vraagteken wordt gevolgd door de cursor. Dit vraagteken geeft aan dat er een commando kan worden ingetypt. Alle commando's bestaan uit twee letters. Op deze manier zijn ze kort maar toch gemakkelijk te onthouden. Er hoeft geen <RETURN> na een commando te worden ingetypt. Als een onbekend commando wordt gegeven verschijnt het vraagteken 'gewoon' weer. Afgezien van de commando's moet alles wat ingetypt wordt, beëindigd worden met <RETURN>-toets. Vrijwel alle commando's kunnen worden afgebroken door <CTRL> <C> in te drukken, d.w.z. <CTRL> tegelijk met <C> in drukken. Als U tijdens het listen de spatiebalk indrukt, stopt het listen om na een volgende druk op de spatiebalk weer verder te gaan.

Als naar een regelnummer wordt gevraagd, kunt u gewoon met een regelnummer antwoorden, maar daarnaast is het ook mogelijk om te antwoorden met een label of een label+getal (of:label-getal), in dat geval wordt het regelnummer bedoeld van de regel waar het betreffende label voor het eerst wordt gedeclareerd (d.w.z. waar het label voor het eerst een waarde krijgt, bijv. met de EQU pseudo-opcode), en dan wordt het getal (indien aanwezig) bij dit regelnummer opgeteld of er van afgetrokken. Dit is erg handig bij grote programma's als U de naam van een subroutine wel weet, maar het regelnummer niet.

1.1 DE EDITOR COMMANDO'S

De editor commando's maken het mogelijk dat u een source kunt intypen en veranderen. Voor de regels in de source gelden een paar regels, die bij het CP-commando worden uitgelegd. Bij het intypen of veranderen van een regel verschijnt een blokje als cursor. Dit betekent dat de editor in 'Overwrite'-mode staat, d.w.z. dat iedere letter die wordt ingetoets over de letter onder de cursor heen gaat. Tevens kan de cursor een liggend streepje zijn ('Insert'-mode), dan wordt elke ingetoetste letter tussengevoegd. Door middel van de <INS>-toets kan worden omgeschakeld tussen 'Insert'- en 'Overwrite'-mode. Hieronder volgt een overzicht van de andere controle-toetsen:

pijltje naar rechts	: indien mogelijk de cursor naar rechts bewegen.
pijltje naar links	: indien mogelijk de cursor naar links bewegen.
<DEL>	: de letter onder de cursor weghalen.
<BS>	: de letter links van de cursor weghalen.

DE EDITOR-COMMANDO'S:

CP (ComPose)

Met dit commando kan een nieuwe source worden ingetypt. Als er nog een source in het geheugen zit, vraagt RF-ASSEMBLER:

```
Code erasure. Proceed (Y/N) ?
```

Als u hierop antwoordt met 'Y' dan wordt het geheugen leeg gemaakt. Als u met 'N' reageert dan wordt het CP-commando niet verder uitgevoerd en verschijnt het vraagteken weer.

Als het geheugen leeg is dan verschijnt er:

```
0001
```

op het beeld. Dit is het regel nummer van de eerste regel. Nu kan een regel worden ingetypt.
Als de eerste regel is ingevoerd verschijnt er:

```
0002
```

op het beeld voor de tweede regel. Dit herhaalt zich voor alle regels tot u <CTRL> <C> hebt ingedrukt.

De regels voor de source zijn als volgt:
1 - Als er een label op een bepaalde regel wordt gedeclareerd, moet de regel niet met een spatie beginnen;
2 - In het andere geval moet de regel minstens met 1 spatie beginnen.
3 - Altijd hoofdletters gebruiken, behalve tussen aanhalingstekens (') of na een puntkomma (;),

Voorbeeld:

```
0001 LOOP DEC BC     ; hier wordt het label LOOP gedeclareerd.
0002      LD A,B     ; hier wordt niets gedeclareerd
0003      OR C
0004      JR NZ,LOOP; hier wordt geen label gedeclareerd,
                      maar wel gebruikt
0005      RET
```

U mag echter ook intypen:

```
0001 LOOP DEC BC
0002  LD A,B
0003  OR C
0004  JR NZ,LOOP
0005  RET
```

Andere regels:

4 - Labels bestaan uit een letter gevolgd door letters of cijfers. De maximum lengte is 8 karakters. Sommige labels mogen niet gebruikt worden:
A,B,C,D,E,H,L,AF,BC,DE,HL,IX,IY,IXH,IXL,IYH,IYL, SP,Z,NZ,C,NC,PE,PO,M,P,I,R.

5 - Constanten beginnen altijd met een cijfer. Hexadecimale constanten worden gevolgd door een 'H', binaire door een 'B' en octale door een 'O'. In plaats een getal kunt U een letter letter tussen quotes (') gebruiken, daarmee wordt dan de ASCII-code van die letter bedoeld. Bijvoorbeeld:

```
CP   'A'        ; is gelijk met CP 65
LD   HL,377O    ; is gelijk met LD HL,255
```

Bovendien kunt U gebruik maken van $. In dat geval wordt voor $ het adres van de instructie ingevuld waar $ wordt gebruikt.
Voorbeeld:

```
LABEL DJNZ LABEL
```

is gelijkwaardig met

```
DJNZ $
```

6 - In rekenkundige uitdrukkingen mag gebruik worden gemaakt van de bewerkingen optellen (+), aftrekken (-), vermenigvuldigen (*), delen (/), logisch AND (&) en logisch OR (!). De prioriteit van de bewerkingen is als volgt: Eerst worden *, / en & uitgevoerd,
daarna +, - en !.
De prioriteiten van deze bewerkingen kan men omzeilen door het gebruik van haakjes. Haakjes mogen maximaal 6 maal binnen elkaar worden gebruikt (genest), anders veschijnt de melding:

```
*** Bad nesting
```

Door het gebruik van haakjes is het soms niet duidelijk wat de betekenis van een ASSEMBLY regel is, b.v.:

```
LD    A,(d)
```

kan betekenen laad A met de constante d, maar tevens laad A met de waarde die op geheugenplaats d staat. In zulke gevallen, zal de assembler (indien mogelijk) aannemen dat de waarde op geheugenplaats d wordt bedoeld.
Voorbeeld:

```
0001        LD    HL,1+2-3+4+OFFSET
0002        LD    (IX+DISPLACE-3),WAARDE+0FFH
0003        LD    A,'T'+1
0004        LD    A,(3)+(4)       ; = LD A,7
0005        LD    A,((3)+(4))     ; = LD A,(7)
0006        XOR   45H
```

Een vuistregel is: gebruik nooit meer haakjes dan nodig is!

Nu volgen een paar voorbeelden van fouten:

```
0001        LD   A,3      ; Het eerste teken had een spatie
                            moeten zijn
0002 LOOP   NOP           ; Het eerste teken mag geen spatie
                            zijn
0003 OTEST  NOP           ; Het eerste teken mag geen cijfer
                            zijn
0004        LD   HL,FFFFH ; FFFFH had OFFFFH moeten zijn
0006        LD   A,'A     ; Er ontbreekt een '
0007        LD   A,3      ; ld had LD moeten zijn
0008 A      NOP           ; A is een verboden label
```

CC (Continue Composition)

De bestaande source wordt hiermee uitgebreid. Op het beeld verschijnt het regelnummer dat na de laatste regel in het geheugen staat. Alles gaat verder als het CP-commando.

ED (EDit)

Er kunnen veranderingen in de bestaande source worden aangebracht. Eerst wordt er gevraagd 'Line#?', nu kunt u het regelnummer van de te veranderen regel intypen (uiteraard daarna <RETURN> indrukken). Vervolgens wordt de betreffende regel afgedrukt.

Nu kunt u de volgende toetsen gebruiken:

```
pijltje naar boven   : de regel ervoor afdrukken,
pijltje naar beneden: de  ,,    erna      ,,     ,
<D> (Delete)         : de regel weghalen,
<I> (Insert)         : regels tussenvoegen, zie (InSert),
<C> (Change)         : de regel veranderen met pijltjes,
                       <INS>, enz..
<CTRL> <C>           : stoppen met het ED-commando.
```

IS (InSert)

Regels kunnen worden tussengevoegd in de bestaande source. Gevraagd wordt: 'Line#?', waarna u kunt aangeven waar een regel moet worden tussengevoegd. Verder verloopt het als het CC-commando.

DL (Delete Line)

Hiermee kan een regel worden weggehaald. Het programma vraagt welke regel. Voorbeeld:

Neem aan dat de source er als volgt uit ziet:

```
0001 ; regel 1
0002 ; regel 2
0003 ; regel 3
```

U wilt nu regel 2 kwijt raken, dat gebeurt als volgt:

```
? DL
Line#? 2
```

Nu hebt u de volgende source:

```
0001 ; regel 1
0002 ; regel 2
```

En dit was de bedoeling.

DM (Delete Multiple lines)

Meerdere regels weghalen, gevraagd wordt:

```
First line#? - hiermee kunt u de eerste  regel aangeven,
Last  line#? - hiermee kunt u de laatste regel aangeven.
```

Vervolgens verwijdert RF-ASSEMBLER de opgegeven regels uit de source.

MB (Move Block)

Een aantal regels verplaatsen binnen de source, gevraagd wordt:

```
First line# ? - eerste  regel van het te verplaatsen blok,
Last  line# ? - laatste regel van het te verplaatsen blok,
Insrt line# ? - de regel waar het blok moet komen te staan.
```

Dit commando kan niet zonder meer worden gebruikt om een aantal regels naar het einde van de source text te verplaatsen. In dat geval zult U eerst een regel aan het einde van de source moeten toevoegen. En dan deze regel als 'Insrt line' gebruiken.
Voorbeeld:

Stel U hebt de volgende source:

```
0001 ; regel 1
0002 ; regel 2
0003 ; regel 3
```

Nu wilt U regel 1 en 2 na regel drie hebben. U voegt nu met het CC-commando regel 4 toe:

```
0004  NOP ; dummy
```

Vervolgens geeft u het MB-commando:

```
? MB
First line# ? 1
Last  line# ? 2
Insrt line# ? 4
```

Hierna krijgt U de volgende source:

```
0001 ; regel 3
0002 ; regel 1
0003 ; regel 2
0004         NOP          ; dummy
```

Omdat regel 4 nu overbodig is geworden, kunt U hem met het DL-commando weghalen.

CS (Change Symbol)

De naam van een label kan worden veranderd. Overal in de source wordt de verandering doorgevoerd. Gevraagd wordt:

```
Original label : - de huidige naam invullen,
New      label : - de nieuwe naam.
```

Een label mag niet de naam van een al bestaand label krijgen, tevens moet het huidige label in de source voorkomen. Voorbeeld:

We willen het label 'ABC' veranderen in 'JANTJE'. Uiteraard komt 'JANTJE' nog niet voor in de source en 'ABC' wel. Dus:

```
? CS
Original label : ABC
New      label : JANTJE
```

Vanaf nu is overal waar 'ABC' in de source voorkwam dit vervangen door 'JANTJE'.

1.2 ASSEMBLER COMMANDO'S

Met de volgende commando's zijn niet gemaakt om de source te veranderen, maar om onder andere de source te bekijken en te assembleren.

LC (List Completely)

De gehele source wordt gelist. Voorbeeld:

```
? LC
0001 ; regel 1
0002 ; regel 2
0003 ; regel 3
?
```

LL (List to the Last line)

De source wordt vanaf een bepaalde regel gelist, RF-ASSEMBLER vraagt vanaf welke regel dit moet gebeuren.

LS (List Symbols)

Bepaalde labels kunnen met hun waarde worden gelist. Met het antwoord op de vraag 'Label mask?' kunt u aangeven welke labels U wilt zien. U kunt gebruik maken van * en ?. (zgn. 'Wildcards') Voorbeelden:

```
A*       - Alle labels die beginnen met een A,
A?ENOOT  - Alle labels die beginnen met een A en
           eindigen op ENOOT.
```

Het gebruik van deze wildcards is vrijwel het zelfde als onder MSX-DOS. Hier mag alleen niet met extensions worden gewerkt (wel bij het DR-commando, zie verderop: DISK COMMANDO'S). Als de waarde van een label bekend is (na het laatste AM-, AS-, LI- of WO-commando) wordt deze naast het label afgebeeld. Voorbeeld:

```
? LS
Label mask? ?A*
LABEL1   F345  LABEL2   0005  JANTJE      AAAA     0000
?
```

Dus alle labels waarvan de tweede letter een A is.

AS (Assemble to Screen)

Vanaf een bepaalde regel kan een assembly listing (inclusief een lijst met de gebruikte labels en hun waarden - zie LS) naar het beeld worden gestuurd. RF-ASSEMBLER vraagt vanaf welke regel dit moet gebeuren. Tevens worden eventuele fouten vermeld.

Hier volgt een lijstje:

```
*** Undefined label         - waarde van 'n label is onbekend,
*** Doubly defined label    - aan een label wordt vaker dan
                              éénmaal een waarde toegekend,
*** Out of range            - een relatieve sprong is te groot.
```

Voorbeeld:

```
? AS
First line#? 2
Pass1
Pass2
0007 00        0002 LABEL     NOP
0008 3E55      0003           LD    A,170

LABEL    0007
?
```

Zoals uit het voorbeeld blijkt, worden na het assembleren ook alle gebruikte labels afgedrukt, evenals bij het LS-commando.

'Pass1' betekent dat RF-ASSEMBLER de waarden van alle labels probeert te bepalen. Hierna verschijnt 'Pass2', wat wil zeggen dat RF-ASSEMBLER de labels overal invult bij het assembleren. Zie verder het AM-, het LI- en het WO-commando.

LI (List Internal errors)

Geeft een lijst van alleen de errors zoals beschreven bij het AS-commando.

AM (Assemble to Memory)

De source wordt rechtstreeks in het geheugen geassembleerd. Als dit echter leidt tot het overschrijven van gereserveerde stukken geheugen dan meldt RF-ASSEMBLER:

```
*** Memory error
    Assembly aborted
```

waarna het assembleren wordt afgebroken. Daarom kan het dus wel eens nodig zijn om een programma ergens anders in het geheugen te assembleren, dan de uiteindelijke bedoeling is.

De assembler kent nog een paar pseudo-opcodes, dit zijn opcodes die geen machinetaal instructie voorstellen:

ORG- (ORiGin)

Geeft aan op welk adres de assembler moet gaan assembleren

Voorbeeld:

```
0001          ORG  100H; Het programma begint op 100H
```

EQU- (EQUals)

Geeft de waarde van een symbol aan (de woorden label en symbol zijn voor de assembler niet verschillend en worden dan ook door elkaar gebruikt).

Voorbeeld:

```
0001 WAARDE   EQU  1234H; WAARDE is gelijk aan 1234H
```

DEFB- (DEFine Byte)

Zet een bepaalde waarde (1 byte) in het geheugen

Voorbeeld:

```
0001          DEFB 5; Zet hier de waarde 5 in 't geheugen
```

DEFW- (DEFine Word)

Als DEFB maar nu worden 2 bytes in het geheugen gezet.

DEFS- (DEFine Space)

Een aantal bytes ruimte reserveren

Voorbeeld:

```
0001          DEFS 2; Laat hier twee bytes ruimte vrij
```

DEFM- (DEFine Message)

Een ascii string in het geheugen zetten

Voorbeeld:

```
0001          DEFM 'Test'; Zet hier het woord test neer
```

END- (END)

Deze pseudo-opcode heeft geen enkele functie. Het is echter zo dat assemblers in het algemeen eisen dat er aan het einde van de source-text een regel staat met end. Het is hoeft bij RF-ASSEMBLER niet, maar het mag. Bovendien wordt er in appendix E gebruik van gemaakt.

Het is aan te bevelen alle voorbeelden even uit te proberen om het even 'in de vingers te krijgen'.

1.3 DISK COMMANDO'S

De volgende commando's hebben betrekking op de disk:

FN (File Name)

De naam waaronder een file wordt geschreven naar disk kan worden veranderd (of gedefinieerd). Eerst wordt de huidige filename afgebeeld, waarna RF-ASSEMBLER naar een nieuwe filename vraagt. Bijvoorbeeld:

```
Old file name : ABCDEFGH.IJK
New file name :
```

Het antwoord moet een geldige filename zijn. Een extension mag ook worden ingetikt, wordt dit niet gedaan, dan maakt RF-ASSEMBLER de extension '.ASM'. voorbeelden van geldige filenames zijn:

```
'A:TEST',
'BESTAND',
'DEMO.'    - Hier is er dus sprake van een lege extension
'TEST.TST'
'FILE1234',
enz...
```

WO (Write Object)

Het geassembleerde programma wordt naar disk geschreven. De extension van de file wordt automatisch .COM, zodat het programma onder MSX-DOS eenvoudig op te starten is. Als er nog geen filename is gedefinieerd, dan wordt er naar gevraagd. Dit gaat op de manier zoals dit bij het FN-commando gebeurt.

Als U gebruik maakt van de pseudo-opcode 'DEFS' dan wordt de ruimte die U hiermee wilt reserveren in het programma opgevuld met nullen. Dit is dus alleen het geval bij het WO-commando.

Bij het gebruik van 'ORG' is ook een waarschuwing op z'n plaats: Het is niet verstandig meerdere malen 'ORG' te gebruiken in een programma; de ruimte die op die manier tussen twee stukken programma kan ontstaan, wordt bij het WO-commando namelijk niet opgevuld. In plaats van

```
ORG   ADDRESS
```

kunt u beter gebruik maken van

```
DEFS ADDRESS-$
```

Op deze manier wordt zo'n ruimte dus wel opgevuld.

WS (Write Source)

De source wordt naar disk geschreven. Als geen filename is gedefinieerd dan wordt er naar gevraagd (Zie het FN-commando). bovendien wordt er de volgende vraag gesteld:

```
Ascii or Code (A/C) ?
```

Als hierop met 'A' wordt geantwoord, dan wordt de source als een normaal leesbare ascii file geschreven, zodat eventuele andere assemblers de file ook kunnen lezen. ASCII-files gebruiken veel meer ruimte op disk, maar bij een fout op de disk kunnen ze eenvoudig worden gecorrigeerd. Bij een Code-file is dit vrijwel onmogelijk, maar deze files gebruiken veel minder ruimte op disk (het kan wel een factor 3 uitmaken!). Bovendien worden ze veel sneller gelezen en geschreven.
Als bij het wegschrijven van een source een oude file wordt overschreven, dan krijgt de oude file de extension '.BAK' (back up). Hierdoor weet U zeker dat 'de vorige versie' van een source nog op disk staat (dus met de extension '.BAK').

RS (Read Source)

Nu kan een source van disk worden gelezen. Altijd vraagt RF-ASSEMBLER naar de filename. Als er nog een source in het geheugen zit dan wordt er gevraagd:

```
Code erasure. Proceed (Y/N) ?
```

Als hierop met 'Y' (Yes) wordt geantwoord, dan wordt de source die nog in het geheugen zit weggehaald. In het andere geval wordt de te lezen source achter de bestaande source in het geheugen gezet. Dan moet de te lezen file echter wel een ASCII-file zijn, anders wordt gereageerd met:

```
*** Bad file mode
```

Bovendien kan er een fout optreden als U probeert sourcefiles in te lezen uit een nieuwere versie van RF-Assembler:

```
*** Bad version
```

Zorg er dus voor dat U Uw garantiekaart ingevuld opstuurt aan de uitgever, zodat U in de toekomst steeds met de laatste updates van deze assembler werkt.

Versie 2.1 kan de sources van versie 1.1 lezen, maar als zo'n source als Code (zie bij 'WS') wordt gelezen dan wordt hij niet zo compact opgeslagen als bij versie 2.1 mogelijk is. Als een source echter als Ascii wordt geladen, gebeurt dit optimaal. Als de source eenmaal met versie 2.1 naar disk is geschreven (als Code) dan blijft deze compacte methode gehandhaafd.

RF-ASSEMBLER herkent zelf of de source een Ascii of Code file is bij het lezen. Als het een Ascii file betreft wordt deze bij het lezen in het geheugen gezet als een Code file. Als de assembler een regel niet kan omzetten, wordt deze op het beeld afgedrukt met een foutmelding. De regel wordt wel in het geheugen opgeslagen, maar dan met een ';' er voor, zodat U er later iets aan kan veranderen.

Bij het lezen kan blijken dat de file te kort is, in dat geval wordt gemeld:

```
*** Unexpected end of file
```

Als dit gebeurt tijdens het lezen van een ASCII-source wordt zoveel mogelijk in het geheugen gezet, bij een Code-source is alles onbruikbaar, De tekst wordt dan ook niet opgeslagen.

DR (DiRectory)

Met dit commando kan worden gekeken welke files er op een disk staan en hoeveel ruimte er nog op de disk vrij is. Gevraagd wordt:

Dir mask :

Hiermee kunt u aangeven welke files moeten worden gelezen. Er kan gebruik worden gemaakt van * en ?. (zgn. Wildcards). Als de vraag niet beantwoord wordt, worden alle files gelist op de disk in de default drive.

Voorbeelden:

```
A:E?EN        - alle files op de disk in drive a:
                met als eerste letter een E en vanaf de 3e
                letter EN,

????????.??? - alle files op de disk in de default drive,
*.*          - alle files op de disk in de default drive.
```

Opmerking:

In de handleiding bij RFASSEMBLER versie 1.1 werd gewaarschuwd voor errormeldingen van MSX-DOS, zoals deze

```
Not ready error reading drive A
Abort, Retry, Ignore?
```

Bij versie 2.1 kan het geen kwaad meer als U voor Abort kiest. Als U daarvoor kiest bent U gewoon weer terug in RF-ASSEMBLER inplaats van dat U terugkeert naar MSX-DOS.

2 - DE DEBUGGER

Om fouten in een machinetaalprogramma op te sporen, kunt u (stukken van) programma's disassembleren, en er met een single-stepper 'doorlopen' waarbij u de mogelijkheid hebt van breakpointing. Tevens zijn er de mogelijkheden om hele blokken van het geheugen (als ascii) te bekijken en om geheugenplaatsen te veranderen. Als U met de debugger werkt is het verstandig de source van Uw programma eerst op disk te zetten. Het is namelijk zo, dat als U gebruik maakt van b.v. het GO-commando, dan kan er iets mis gaan als er bijvoorbeeld een fout zit in het te debuggen programma! De debugger werkt over het algemeen met hexadecimale getallen en heeft de volgende commando's:

MD (Memory Dump)

Een stuk geheugen wordt afgebeeld, zowel hexadecimaal als ascii. RF-ASSEMBLER vraagt Address ? om te weten vanaf welk adres het commando alles moet afdrukken. Er verschijnt dan een regel met de inhoud van 16 geheugenplaatsen. Door een willekeurige toets in te drukken (behalve <CTRL> <C>) worden de volgende 16 geheugenplaatsen afgedrukt.

MM (Memory Modify)

De geheugeninhoud kan worden veranderd. Er wordt gevraagd welk adres als eerste moet worden veranderd. Daarna verschijnt steeds het adres met de bijbehorende geheugeninhoud. Op dat moment kan men een nieuwe waarde intypen of gewoon <ENTER> intoetsen om de inhoud niet te veranderen. Hierna herhaalt alles zich met het volgende adres. Voorbeeld:

```
? MM
Address (hex) ? 8000
8000 55
8001 00 44
8002 34 1
```

DS (DisaSsemble)

Een stuk geheugen kan worden gedisassembleerd. Er wordt gevraagd vanaf welk adres dit moet gebeuren. Door steeds een willekeurige toets in te drukken (behalve <CTRL> <C>) wordt het volgende adres gedisassembleerd.

AI (Assemble Immediately)

Tijdens het testen van een geassembleerd programma komt het wel eens voor dat het handig is een extra stukje machinetaal in het geheugen te zetten, zonder de source te moeten veranderen. Dit kan eenvoudig met het AI-commando. Na het intikken van 'AI' vraagt RF-ASSEMBLER U naar het adres ('Address (hex) ?') waar het extra stukje programma moet staan. Vervolgens wordt de instructie die op dat moment op dat adres staat gedisassembleerd. U kunt hier nu een andere instructie neerzetten, gewoon door het intikken van een assembly-taal instructie, voorbeeld:

```
? AI
Address (hex) ? 8000
8000 00        NOP         LD A,(9001H)
8003 00        NOP         NOP
```

Dit herhaalt zich tot U <CTRL> <C> indrukt. Bij dit commando kunt U alle dingen intypen die bij het CP-commando ook kunt intypen, behalve regels waarin labels worden gebruikt. Het is dus wel belangrijk bij dit commando als U een adres hexadecimaal bedoelt, dat U er dan ook een 'H' achter zet.

BK (BreaKpoint)

Als U bij het debuggen van een programma ervan overtuigd bent dat een bepaald stuk prima in orde is, dan is het zinloos om daar Single-Step doorheen te moeten. In zo'n geval kunt U een breakpoint aan het eind van zo'n stuk programma neerzetten. Dan kunt u met behulp van het GO-commando de controle overdragen aan het programma. Als het programma dan bij het breakpoint komt, belandt u weer in RF-ASSEMBLER. Er kunnen maximaal 8 breakpoints worden gedefinieerd. Bij het gebruik van dit commando worden eerst de reeds gedefinieerde breakpoints afgedrukt. Als er nog een breakpoint bij kan (maximaal 8) wordt er naar gevraagd:

Address (Hex) ?

Het antwoord hierop moet het adres zijn waar U een breakpoint wilt neerzetten.

CB (Clear Breakpoint)

Met dit commando kunnen breakpoints worden weggehaald. Er verschijnt nu een lijstje met de bestaande breakpoints. (Als er geen breakpoints zijn gebeurt er niets.) Bijvoorbeeld:

```
 Breakpoints :
1 9000
2 9001
```

De breakpoints zijn genummerd, dus breakpoint no. 1 zit nu op adres 9000. Vervolgens wordt er gevraagd:

Breakpoint no. ?

Nu kunt u het nummer invoeren, van het breakpoint dat weg moet. In plaats van een nummer kunt U ook met een 'A' antwoorden, dit staat voor 'All'. Als U dat doet worden alle breakpoints veranderd.

GO (GO)

Om de controle over te dragen aan het programma dat U aan het debuggen bent, moet U dit commando gebruiken. Omdat dit commando bij verkeerd gebruik nogal funest kan zijn, wordt eerst gevraagd:

Go. Proceed (Y/N)?

Als hierop met 'Y' wordt geantwoord, wordt het te debuggen programma programma uitgevoerd tot het een breakpoint 'tegen komt'.

SP (SteP)

Dit is de single-stepper. Op deze manier kan men het programma instructie na instructie uitvoeren, waarbij de registers allemaal worden afgebeeld. Het adres waar de single-stepper begint staat in het PC-register (Program Counter). Dit register moet eerst de goede waarde hebben, dat kan met het RG-commando. Door steeds op de spatiebalk te drukken wordt de

instructie die op het beeld staat uitgevoerd. Als een X (eXecute) wordt ingedrukt, worden alle CALL's (en RST's) in 1 keer volledig uitgevoerd. Door een G (Go) in te drukken, wordt het GO-commando uitgevoerd. Na <CTRL> <C> (om bijvoorbeeld een geheugenplaats of een register te veranderen) kan men door het nogmaals intypen van 'SP' gewoon verder gaan.

RG (ReGister)

Een register kan een waarde krijgen. Eerst worden de huidige waarden van de registers en de status van de interrupt afgebeeld. bijvoorbeeld:

```
 F : NC,NZ,PO,P,EI
AF 0000 BC 0000 DE 0000 HL 0000 IX 0000 IY 0000
A' 0000 B' 0000 D' 0000 H' 0000 SP D505 PC 0000
```

Zoals blijkt wordt het F (Flags) register tweemaal afgebeeld. Op de eerste regel worden de flags apart afgebeeld, bovendien wordt de interrupt status hier ook afgebeeld. Welk register een andere waarde krijgt hangt af van het antwoord op de vraag

```
Register :
```

De volgende antwoorden zijn mogelijk: AF, BC, DE, HL, IX, IY, A'(=AF'), B'(=BC'), D'(=DE'), H'(=HL'), SP, PC, A, B, C, D, E H, L, IXH, IYH, IXL, IYL en F.

Het antwoord op de vraag

```
Value    :
```

Bepaalt de waarde van een register. Bij het F (Flags) register moet u geen getal invoeren, maar met een (of meerdere, gescheiden door een komma) van de volgende 'Flags':

```
NC - De carry-flag wordt 0        C  - De carry-flag wordt 1
NZ - De zero-flag  wordt 0        Z  - De zero-flag  wordt 1
PO - De P/V-flag   wordt 0        PE - De P/V-flag   wordt 1
P -  De sign-flag  wordt 0        M  - De sign-flag  wordt 1
```

Bovendien kan men ook de interrupt-status beinvloeden. Dit wordt mogelijk gemaakt doordat U naast de hierboven genoemde flags ook het volgende kunt intypen:

DI - Disable interrupt E - Enable interrupt

Een correct antwoord zou dus zijn:

```
NC,DI,PO,Z
```

CV (ConVert)

Soms is het handig om bijvoorbeeld een hexadecimaal getal om te zetten in een decimaal, octaal of binair getal. Daar is dit commando voor. Eerst wordt gevraagd:

```
Value  :
```

Hierop kunt u antwoorden met een willekeurig getal: binair, octaal, decimaal of hexadecimaal. Dit getal wordt vervolgens afgedrukt als hexidecimaal, decimaal, octaal en binair getal. Bijvoorbeeld:

```
 Value     : 10101010B
 OAAH     170       252O     10101010B
```

3 - OVERIGE COMMANDO'S

Er zijn nog een paar commando's over die niet bij de assembler en niet bij de debugger horen. Ze hebben betrekking op het complete functioneren van RF-ASSEMBLER. Het zijn:

FM (Free Memory)

RF-ASSEMBLER drukt het geheugen af wat vrij is, hier kan worden geassembleerd zonder dat er de melding

```
*** Memory error
```

verschijnt. Ook wordt afgedrukt hoeveel (verschillende) labels er in gebruik zijn. Het maximum is 1200 voor computers met een MEMORY-MAPPER en 800 voor computers zonder. voorbeeld:

```
? FM
Free mem :  9964 (B201-D505)
# Labels :   943 (Max: 1200)
```

In dit voorbeeld zijn de geheugenplaatsen van 0B201H tot 0D505H vrij om te gebruiken. Het maximum aantal labels is nu 1200, en er zijn totaal 943.

PT (PrinTer)

Hiermee kan de printer aan of uit worden geschakeld. Als de printer aan staat (Printer on), wil dat zeggen dat alles wat op het beeld verschijnt ook wordt afgedrukt. Als de printer geblokkeerd is, en daardoor ook RF-ASSEMBLER, dan kan dit met <CTRL> <STOP> worden opgelost.

QT (QuiT)

Met dit commando keert u terug naar MSX-DOS. Voordat dit echter wordt uitgevoerd wordt er gevraagd:

```
Quit. Proceed (Y/N)?
```

Als U hierop met 'Y' antwoordt, keert U inderdaad naar MSX-DOS terug. Zorg wel dat U de source enz. naar disk hebt geschreven als U dit commando gebruikt, want anders bent U alles kwijt!

APPENDIX A

Foutmeldingen van RF-ASSEMBLER

RF-ASSEMBLER kan de volgende foutmeldingen geven:

*** Bad

- Bij het beantwoorden van een vraag (bv.'Line#?') is een verkeerd antwoord gegeven.

*** Bad file mode

- Bij het mergen - RS-commando blijkt de te mergen file geen ASCII-file te zijn.

*** Bad label

- Bij het inlezen van een source regel is een illegaal gedeclareerd label ontdekt.

*** Bad nesting

- Bij het inlezen van een soucre regel zijn in een expressie te veel haakjes binnen elkaar geplaatst (genest)

*** Bad opcode

- Bij het inlezen van een source regel is een onbekende opcode ontdekt.

*** Bad operand

- Bij het inlezen van een source regel is een onbekende operand ontdekt.

*** Bad version

- Bij het lezen van een Code-file (RS-Commando) blijkt deze te zijn geschreven door een nieuwere versie van RF-Assembler.

*** Disk full

- Bij het schrijven naar disk blijkt deze vol te zijn.

*** Double defined label

- Bij het assembleren (Pass 1) blijkt een label twee maal gedeclareerd te zijn.

*** Expression too long

- Bij het inlezen van een source-regel is een te lange expressie ontdekt.

*** File not found

- Bij het 'DR'-commando of het 'RS'-commando is de genoemde file niet gevonden.

*** Memory error
Assembly aborted

- Bij het assembleren naar het geheugen ('AM'-commando) blijkt dat er gereserveerd geheugen wordt gebruikt. Het aasembleren wordt onmiddellijk afgebroken.

*** Memory full or too many labels

- Het beschikbare geheugen is vol of er zijn te veel labels

*** Missing label declaration

- Bij het inlezen van een source-regel is ontdekt dat in een 'EQU'-regel geen label is gedeclareerd.

*** No label declaration allowed

- Bij het inlezen van een source-regel is ontdekt dat in een 'ORG'-regel een label is gedeclareerd, wat niet is toegestaan.

*** No source file

- Bij het lezen van een Code-File ('RS'-commando) blijkt deze niet door een RF-Assembler te zijn geschreven.

*** Out of range

- Bij het assembleren (Pass 2) blijkt een te grote sprong (bij JR of DJNZ) te moeten worden vertaald.

*** Printing aborted

- De printer wordt 'uitgeschakeld' door het indrukken van: <CTRL> <C>.

*** Undefined label

- Bij het assembleren blijkt in de source te worden verwezen naar een niet gedeclareerd label.

*** Unexpected end of file

- Bij het lezen van een file ('RS'-commando) blijkt de file niet volledig op disk te staan.
  Als het om een code-file gaat, wordt de code niet geaccepteerd.

APPENDIX B

De commando's van RF-ASSEMBLER

Hier volgt een lijst met de commando's van RF-ASSEMBLER met hun betekenis:

Editor commando's:

CC (Continue Composition)	- De source uitbreiden
CP (ComPose)	- Een (nieuwe) source maken
DL (DeLete)	- Een regel weghalen
DM (Delete Multiple lines)	- Meerdere regels weghalen
ED (EDit)	- Regels veranderen
IS (InSert)	- Regels tussenvoegen
MB (Move Block)	- Regels verplaatsen

Assembler commando's:

AS (Assemble to Screen)	- Een assembly listing maken
AM (Assemble to Memory)	- Naar het geheugen assembleren
LC (List Completely)	- Een listing van de source
LI (List Internal errors)	- Errors aangeven
LL (List to Last line)	- Een listing tot de laatste regel
LS (List Symbols)	- De labels (+inhoud) weergeven

Disk commando's:

DR (DiRectory)	- Directoryinhoud weergeven
FN (FileName)	- Een filename definieren
RS (Read Source)	- Een source lezen
WO (Write Object)	- Naar disk assembleren
WS (Write Source)	- Een source schrijven

Debugger commando's:

AI (Assemble Immediate)	- Rechtstreeks assembly instructie in het geheugen zetten
BK (BreaKpoint)	- Een breakpoint definieren
CB (Clear Breakpoint)	- Een breakpoint weghalen
CV (ConVert)	- Getallen omzetten
DS (DiSassemble)	- Geheugen disassembleren
GO (Go)	- Een programma uitvoeren
MD (Memory Dump)	- Geheugeninhoud weergeven
MM (Memory Modify)	- Geheugeninhoud veranderen
RG (ReGister)	- De waarde van een register veranderen
SP (SteP)	- Machine instructies een voor een uitvoeren

Overige commando's

FM (Free Memory)	- Vrij geheugen aangeven
PT (PrinTer)	- De printer aan/uit schakelen
QT (QuiT)	- Terug naar MSX-DOS

APPENDIX C

Welke Z-80 intructies kent RF-ASSEMBLER

RF-ASSEMBLER kent alle officiele Z-80 instructies. Bovendien herkent de assembler nog een aantal 'extra' instructies. Deze komen niet in de specificaties van de Z-80 voor, maar ze kunnen wel degelijk worden gebruikt. Het betreft de instructies die werken met de IXH, IXL, IYH en IYL registers. Hiermee wordt bedoeld:

```
IXH/IYH : de hoogste 8 bits van het IX/IY register,
IXL/IYL : de laagste 8 bits van het IX/IY register.
```

Voorbeelden:

```
LD   A,IXH,
INC  IXH,
ADD  A,IYL,
LD   IYH,5.
```

Behalve bovenstaande uitbreidingen, herkent RF-ASSEMBLER versie 2.1 de SLL instructie. Deze instructie is gelijkwaardig met:

```
SCF
RL  r       ; r is een willekeurig 8 bits register
```

Over het algemeen, werken deze 'extra' instructies prima, maar ze worden niet gegarandeerd door Zilog. Het kan dus geen kwaad deze instructies op Uw eigen computer te gebruiken, maar het is niet zeker of ze ook op andere computers werken.

Een lijst met alle Z-80 instructies die RF-ASSEMBLER kent:

```
ADC  A,s        IND             LDDR
ADC  HL,ss      INDR            LDI
ADD  A,s        INI             LDIR
ADD  HL,ss      INIR            NEG
ADD  IX,pp      JP   (HL)       NOP
ADD  IY,rr      JP   (IX)       OR   s
AND  s          JP   (IY)       OTDR
BIT  b,m        JP   cc,nn      OTIR
```

CALL	cc,nn	JP	nn	OUT	(C),r
CALL	nn,	JR	C,e	OUT	(n),A
CCF		JR	NC,e	OUTD	
CP	s	JR	NZ,e	OUTI	
CPD		JR	Z,e	POP	IX
CPDR		JR	e	POP	IY
CPI		LD	(BC),A	POP	qq
CPIR		LD	(DE),A	PUSH	IX
CPL		LD	(HL),n	PUSH	IY
DAA		LD	(HL),r	PUSH	qq
DEC	IX	LD	(IX+d),n	RES	b,m
DEC	IY	LD	(IX+d),r	RET	
DEC	m	LD	(IY+d),n	RET	cc
DEC	ss	LD	(IY+d),r	RETI	
DEC	x	LD	(nn),A	RETN	
DEFB	n	LD	(nn),IX	RL	m
DEFM	'cs'	LD	(nn),IY	RLA	
DEFS	nn	LD	(nn),ss	RLC	m
DEFW	nn	LD	A,(BC)	RLCA	
DI		LD	A,(DE)	RLD	
DJNZ	e	LD	A,I	RR	m
EI		LD	A,(nn)	RRA	
END		LD	A,R	RRC	m
EQU	nn	LD	I,A	RRCA	
EX	(SP),HL	LD	IX,(nn)	RRD	
EX	(SP),IX	LD	IX,nn	RST	p
EX	(SP),IY	LD	IY,(nn)	SBC	A,s
EX	AF,AF'	LD	IY,nn	SBC	HL,ss
EX	DE,HL	LD	R,A	SCF	
EXX		LD	r,m	SET	b,m
HALT		LD	r,n	SLA	m
IM	0	LD	SP,HL	SRA	m
IM	1	LD	SP,IX	SLL	m
IM	2	LD	SP,IY	SRL	m
IN	A,(n)	LD	ss,(nn)	SUB	s
IN	r,(C)	LD	ss,nn	XOR	s
INC	IX	LD	u,x		
INC	IY	LD	x,n		
INC	m	LD	x,u		
INC	ss	LD	x,x'		
INC	x	LDD			

Met de volgende betekenis:

```
b  : Een getal van 0 t/m 7
d  : Een getal van 0 t/m 255
e  : Een getal van -128 t/m 127            (*)
n  : Een getal van 0 t/m 255
nn : Een 2-byte getal
p  : Een van de waarden 0,8,10H,18H,20H,
     28H,30H of 38H                        (**)
r  : A,B,C,D,E,H of L
m  : r of (HL), (IX+d), (IY+d)
u  : A,B,C,D,E
x  : IXH,IXL,IYH,IYL
x' : IXH,IXL,IYH,IYL                       (***)
s  : r,n,(HL),(IX+d)
cs : Een ascii-string bv:'AapNootMies'
cc : Een van de condities NZ,Z,NC,C,PO,PE,P,M
pp : BC,DE,IX,SP
qq : AF,BC,DE,HL
rr : BC,DE,IY,SP
ss : BC,DE,HL,SP
```

(*) : Er wordt '*** Out of range' gemeld, als e niet voldoet.
(**) : Adressen worden naar beneden afgerond.
(***) : Als x een deel van IX is, moet x' dat ook zijn,
als x een deel van IY is, moet x' dat ook zijn.

APPENDIX D

Het aanroepen van BIOS-routines onder MSX-DOS

Onder MSX-DOS zijn de BIOS-routines niet zondermeer te gebruiken. Dit komt dat de BIOS-ROM dan is uitgeschakeld. Dit is nogal hinderlijk bij het testen van programma's met behulp van de debugger, omdat RF-ASSEMBLER onder MSX-DOS werkt. De oplossing van dit probleem is echter vrij eenvoudig; U kunt namelijk gebruik maken van een zogenaamde INTER-SLOT-CALL. Hiermee kunnen programma's vanuit een bepaald slot subroutines in andere slots aanroepen. Een INTER-SLOT-CALL ziet er als volgt uit:

```
RST  30H       ; De subroutine bij 30H voert de
DEFB SLOTID    ;       INTER-SLOT-CALL uit
DEFW ADDRESS   ; Het adres van de routine
```

Dit is ongeveer gelijk aan:

```
CALL ADDRESS
```

Maar dan dus naar een programma (eventueel) in een ander slot.

Met SLOTID wordt een byte bedoeld, met de volgende betekenis (per bit):

```
FxxxSSPP (bit 7....bit 0)
```

Waarbij:

```
F  : F=1 als secundair slot, anders 0
SS : secundair slotnummer
PP : primair slotnummer
```

Omdat bij MSX-computers de BIOS-ROM in primair slot 0 zit, ziet zo'n INTER-SLOT-CALL er dus als volgt uit:

```
RST  30H
DEFB 0         ; Immers primair slot 0!
DEFW ADDRESS
```

Een voorbeeld:

Als we vanuit een programma een letter naar het scherm sturen, kunnen we daarvoor de BIOS-routine CHPUT gebruiken. Normaal staat er dan voor in het programma:

```
CHPUT  EQU 0A2H
```

Dit wordt nu:

```
CHPUT  RST  30H
       DEFB 0      ; De slotaanduiding, dus primair slot 0
       DEFW 0A2H   ; Het adres in de BIOS-ROM
       RET
```

Nu kunnen we CHPUT ook vanuit MSX-DOS aanroepen. Dit dus vooral om een programma te testen. Deze methode werkt tevens onder BASIC. Dus als het programma, als het klaar is, onder BASIC moet werken, hoeven we het niet weer te veranderen. Dat kan natuurlijk geen kwaad, maar het is niet zo zinvol.

Als we een routine in de EXTENDED ROM (MSX2) willen aanroepen, kunnen we dit niet doen met de BIOS-routine EXTROM, zo is gebleken. Er is echter een andere manier; namelijk via adres 1CH (CALSLT). Dit werkt zowel onder MSX-DOS als onder BASIC. Het gaat als volgt:

```
LD   IX,ADDRESS  ; IX  = adres van routine
LD   IY,(0FAF7H) ; IYH = Slotaanduiding extended ROM
JP   1CH         ; UITVOEREN
```

Voorbeeld:

```
PAINT LD   IX,69H       ; Adres van paint
      LD   IY,(0FAF7H)
      JP   1CH
```

Nogmaals: beide genoemde methoden, zowel voor BIOS als extended ROM werken zowel onder BASIC als onder MSX-DOS. Bij de eerste methode (met RST 30H) moet echter nog iets vermeld worden: Nooit met de single-stepper door een routine gaan zoals:

```
LABEL  RST  30H
       DEFB SLOTID
       DEFW ADDRESS
       RET
```

Dit werkt niet! Zo'n subroutine moet in zijn geheel worden uitgevoerd! Dit is niet zozeer een fout van RF-ASSEMBLER als wel het gevolg van de 'vreemde' opbouw van de INTER-SLOT-CALL.

APPENDIX E

Het maken van 'BLOAD'-files

De programma's die onder MSX-DOS werken wijken sterk af van de 'BLOAD'-files die onder BASIC werken:

- MSX-DOS-programma's beginnen altijd op adres 100H. Omdat dit vaststaat wordt geen informatie in de file gestopt om dit bij te houden.
- BLOAD-programma's kunnen op willekeurige adressen beginnen. Bovendien kan het executie-adres worden meegegeven. Deze informatie wordt dus ook in de file opgeslagen. Om BLOAD-programma's te herkennen heeft het eerste byte van zo'n file altijd de waarde 0FEH.

Ondanks deze verschillen is het geen enkel probleem om met RF-ASSEMBLER een BLOAD-file te maken. Het begin van zo'n file moet er als volgt uit zien:

```
DEFB 0FEH
DEFW BEGIN   ; Het beginadres
DEFW END     ; Het eindadres
DEFW ENTRY   ; Het executie-adres
```

Een voorbeeld:

```
;
; voorbeeldprogramma
;
        DEFB 0FEH     ; Deze regels tijdens
        DEFW BEGIN    ; debuggen niet in de
        DEFW END      ; source opnemen,
        DEFW ENTRY    ; vanwege Memory error!
;                       Alleen in uiteindelijke
;                       versie.
;
        ORG  9000H
```

```
;
; programmatekst
;
BEGIN    JP   ENTRY
;
CHPUT    RST  30H      ; Letter naar beeld
         DEFB 0        ; via INTER-SLOT-CALL
         DEFW 0A2H
         RET
;
; hoofdprogramma
;
ENTRY    LD   A,'a'    ; Executieadres
         CALL CHPUT
         RET
;
END      END
```

Bovenstaand programma kan met het BLOAD-commando onder BASIC worden opgestart. Het drukt dan een 'a' op het beeld af.
Opm: In bovenstaand voorbeeld wordt handig gebruik gemaakt van de pseudo-opcode 'END'.